VERSION MANAGEMENT TOOL

MICROFICHE APPENDIX

A source code listing of a preferred embodiment of the invention is appended in the form of 1240 pages recorded on microfiche.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to the field of software development tools and more particularly relates to a system for maintaining control of various versions of a file and for merging different versions of the same file.

2. Description of the Relevant Art

An important type of software development tool manages multiple versions of a common text file. e.g., a program. Typically, this function is required in a software development environment where programs are revised frequently, often by different people, and where previous versions must be preserved. Also, it may be required to merge the changes introduced into independent versions of a common file.

One approach to developing such a management system is described in an article by Tichy entitled "RCS-A System for Version Control", Software-Practice and Experience. John Wiley & Sons, N.Y., 1985, pp. 637–654. That article describes common methods for preserving every version of a file and for merging versions of the same file.

Turning first to the method of preserving versions, the delta method is described in the abovereferenced article. A delta is a series of edit commands that change one version into another. A delta may be a forward delta, for changing a given version into the immediately following version, or a reverse delta, for changing a given version into an immediately preceding version.

A reverse delta system is often preferred because experience shows that often the most frequently required version is the last version created. In a reverse delta system, the last version is stored intact and may be immediately accessed. Text files of previous versions must be created by successively applying reverse deltas to the last version created.

A significant limitation in using the delta system to create a text file of a version separated from the last version by several intermediate versions is that each intermediate version must be created and changed by the appropriate delta. This iterative process limits the speed of accessing intermediate versions.

A given file may develop along a single path where each version evolves from the immediately preceding version. Often however, development of a file may proceed along several independent paths. For example. a first programmer may check out a given file and proceed to create several versions along a first path. A second programmer may check out the same given file and proceed to create several versions along a second path. The changes introduced along the first and second paths may be completely independent, e.g.. the two programmers might not talk to each other and have different development goals. The paths are related since they diverge from the same given file.

In some cases, it may be desirable to merge two versions created along independent paths to form a resulting version incorporating the changes introduced along both paths. For example, referring to the example above, the first programmer may be improving a first aspect of the given program and the second programmer may be improving a second aspect. When they have completed their tasks, a new program improved in both aspects may be created by merging versions from the two paths.

A problem in merging occurs when the same line of the given program is changed in both versions to be merged. It is possible that the changes made will conflict and not be compilable. Accordingly, merge programs have been developed including rules for processing lines changed in both versions.

In many existing systems, the lines included in a given version are identified by line numbers. During a merge, the line numbers in the versions to be merged are compared and lines identified by specified line numbers are included in a resulting version. However, often lines will be duplicated in the resulting version because identical lines will have different line numbers in the versions to be merged.

Accordingly, there is a need for improved software development tools having efficient systems for preserving development versions of a text file, creating desired versions, and merging versions developed along independent paths.

SUMMARY OF THE INVENTION

The present invention is a version management tool having improved systems for preserving all versions, creating any desired version, and merging versions developed from a common file along independent paths.

In a preferred embodiment, the fundamental unit for editing is a complete line of text. Thus, to change a character in a given line of text requires that the given line be deleted and replaced by a new line having the desired character changed. Every line active (included) in any version of a given file is included in an indexed line file and tagged by a unique line identifier (ULI). The history of the status of each line in the various versions is recorded in a variant history file.

According to one aspect of the invention, the variant history file is an ordered set of records with each record including a ULI. a version number, and a status flag. In one embodiment, if the value of the status flag is (A) then the line identified by the ULI becomes active in the version identified by the version number included in the record, if the value of the status flag is (D) then the line identified by the ULI is deleted in the version identified by the version number included in the record, and, if the value of the status flag is (R) then the line identified by the ULI is replaced in the version identified by the version number in the record.

According to a further aspect of the invention, the text file of a desired version is created by searching the variant history records to identify ULIs of lines active in the desired version. The lines identified are retrieved from the line file and included in the created text file. The search is facilitated by ordering the records in the variant history file so that a record indicating that a ULI is replaced immediately follows the record indicating that the ULI became active. These records are processed as a pair and the version numbers compared to the version number of the desired version. If the ULI becomes active in a version created before the desired version and is deleted or replaced in a version following the given version then the ULI is active in the desired